LSU EE 4702-1 **Homework 1** **Due: 19 September 2014**

Some of the effort for this homework assignment is in learning to use the various pieces of software, such as a text editor. Those less familiar with Linux software development procedures might seek out a more knowledgeable classmate to minimize frustration and wasted time.

Problem 0: Follow the instruction on the `http://www.ece.lsu.edu/koppel/gpup/proc.html` page for account setup and programming homework work flow. Compile and run the homework code unmodified. It should show a line of balls drop to the platform. *Promptly report any problems.*

Use the arrow keys, PageUp, and PageDown to move around the scene. Use key `h` to toggle between the first (head) ball being locked in place and free. Use key `t` to do the same for the last (tail) ball. Press (lower-case) `b` and then use the arrow and page keys to move the first ball around. Press `l` to move the light around and `e` to move the eye. Press `1` to set up scene 1, press `2` to set up scene 2.

Look at the comments in the file `hw01.cc` for documentation on other keys. One fun thing to do is to lock both the first and last ball, move the head ball until the spring is stretched tight, then release one of the balls. Press `p` to pause, then the space bar to single step. *Note: There is nothing to turn in for this first problem.*

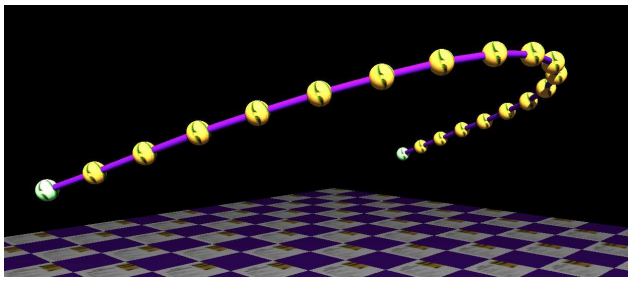
Problem 1: Pressing `1` sets up scene 1 (by calling `ball_setup_1`), the pendulum, and pressing `2` sets up scene 2. In an unmodified assignment setting up scene 2 only freezes the head and tail balls in place, it makes no other changes.

Modify scene 2 (look for code `ball_setup_2`) so that it places the balls evenly spaced in a straight line. The first ball (`ball[0]`) should be placed at position `first_ball` (which is declared in the routine), the last ball (`ball[chain_length-1]`) should be placed at position `last_ball`, and the other balls should be spaced evenly between them. The total number of balls is `chain_length`.

The modified code should make good use of the coordinate classes. For example, DON'T set `x`, `y`, and `y` members individually.

The mathematics for this problem is fairly simple: you need to find a vector pointing from `first_ball` to `last_ball` whose length is the distance between two adjacent balls. Look at `ball_setup_1` to see how balls are initialized.

Problem 2: Pressing `w` toggles a twirl option on and off. In the unmodified code that just toggles variable `opt_twirl` between `true` and `false` and changes an indicator on the upper-left area of the screen. Modify the code so that when `opt_twirl` is true forces are applied to the balls that will cause them to move around an axis defined by the first and last balls. (See the illustration to the right.)



The magnitude of the force should be 1 and the direction should be orthogonal to the axis defined by the first and last ball and the shortest line from the ball to that axis. (As though the ball were embedded in a rotating cylinder with the first and last balls on the axis.) The solution to this problem should go in the `time_step_cpu` routine.

The mathematics for this problem is a little more complex, but still basic. Write an equation for the line connecting the head and tail (first and last) balls (the axis), say $S = B_0 + tv$, where

B_0 is the first ball and v is a vector pointing towards the last ball. If we are choosing a force for ball B, we need to find the point on the axis closest to B. For such a point $\overrightarrow{SB}$ will be orthogonal to v. One can use a property of the dot product to solve for S and another operation to find the force direction, a vector orthogonal to both v and $\overrightarrow{SB}$.